Displaying Web Documents through Negotiation and Dynamic Rendering

by
Sharad Gang

Department of Computer Science & Engineering

Indian Institute of Technology Kanpur

January, 1998

Displaying Web Documents through Negotiation and Dynamic Rendering

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Sharad Gang

to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur
January, 1998

Certificate

Certified that the work contained in the thesis entitled "*Displaying Web Documents through Negotiation and Dynamic Rendering*", by Mr.*Sharad Gang*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

T V Prabhakar

(Dr. T.V. Prabhakar)
Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

January, 1998

Abstract

Web authors create documents in a variety of languages using a variety of character sets and fonts. It is not possible for the viewer of the document to have all those fonts and character sets present on his system. There is a need to allow the viewer to view the documents written using fonts and scripts other than those present on the viewer's (client) system. A number of solutions for this have been proposed. But, all the solutions require the viewer to either download the fonts and install them on his/her system or install some software on his system to help in the process. For a truly portable solution, the client need not specially install any fonts or software on his system. We aim to provide this kind of portable solution. Though we have provided a solution for Devnagari script, the approach can be easily modified for other scripts.

Acknowledgments

I would like to express my deep sense of gratitude to my thesis supervisor, Dr. T. V. Prabhakar for his expert guidance and encouragement throughout this thesis work. Inspite of his hectic schedule he was always ready to listen to my problems and take steps to remedy them. I am indebted to him for all the invaluable suggestions that he provided throughout the course of this project.

Special thanks are due to Atul Dobhal who helped me in installing Windows NT and also recovering it from crashes. I would also like to thank K.V. Vihari, K.S. Shyamsunder and V. Rajesh for their technical help and guidance.

The faculty and staff of the Computer Science department extended their help whenever required and I duly acknowledge this fact.

I am grateful to my parents and brothers for being a constant source of motivation.

Last but not the least, I acknowledge the help and motivation that my colleagues have, knowingly or unknowingly, provided me in my endeavour.

Contents

Chapter 1

Introduction

A wide number of languages are spoken by human beings in this world. Most of these people would prefer to have information presented in a language that they understand. The World Wide Web being the largest repository of information should present this information, whereby maximum people can take benefit of it. However, despite its name, it was not originally conceived as a system able to encompass the whole world: the main limitation being in the area of text representation. People create web documents when they have something to say, information to provide, a message to preach, feeling to express, a product to sell. But, unfortunately they are not able to cater to the widest of the masses. This is primarily because of the sweeping dominance of English and western-european languages on the Web. Most people use English as a second language, and their grasp of it may be quite rudimentary. For in-depth discussions, almost everyone tends to fall back on his or her native language. If the Internet does not allow multilingual conversations, its role as a facilitator of international communication will be severely limited. The Web must be enhanced to meets the needs of the global community. In this thesis, we aim to provide one such solution for Devnagari language.

1.1 Multilingualism On WWW

While the Internet is becoming an ever more global network extending to more and more countries and reaching a constantly increasing number of people. the standards, protocols and software on which it is based are beginning to show limitations in terms of the increased number of user languages. The Hypertext Markup Language (HTML) is a markup language used to create hypertext documents that are platform independent. HTML is used to format the major part of the Web's content. The application of HTML is seriously restricted by its reliance on the Latin-1 (ISO-8859 -1) coded character set. Latin-1 is an 8-bit encoding and permits a maximum of 256 characters. It is only sufficient for Western European languages. Having a large character set is a basic element of Multilingualism; i.e., processing several languages with very different alphabets simultaneously. It is for this reason, the newer version(4.0) of HTML uses UNICODE as its document character set.

1.1.1 UNICODE

The Unicode [25, 16] Character Standard is a character coding system designed to support the interchange, processing, and display of the written texts of the diverse languages of the modern world. UNICODE (ISO 10646 BMP) is a large character set that includes most of the world languages. Unicode is a 16-bits encoding. This permits over 65,000 characters. The current version (2.0) of Unicode standard contains 38,885 distinct coded characters derived from 25 supported scripts. These characters cover the principal written languages of the America, Europe, the Middle East. Africa, India, Asia and Pacifica. Using Unicode as a document character set allows one to use multiple languages in a single document. This means one can include a Sanskrit quotation in a Tamil document unambiguously.

Unicode is the first plane of ISO 10646; this plane is also called BMP (Basic Multilingual Plane) or Plane Zero. ISO 10646 is a 32-bits encoding. It is divided into 32,000 planes, each with 64,000 characters capacity. This permits 2,080 million characters. This form is also called UCS-4, Universal Character Set 4-bytes. Only the first plane (Unicode) is in use.

1.1.2 Character Encoding

The 16 bits are both the force and the weakness of Unicode. The force because one can represent all the characters; the weakness because it is an overkill for English. Without compression, it duplicates the disk space and transmission by two. The top 8 bits are not needed for English; i.e., they are set to zero. For this reason, a suitable character encoding mechanism should be used. A character encoding[19] represents some subset of the document character set. Character encodings such as ISO-8859-1 (encodes most Western European languages), ISO-8859-5 (which supports Cyrillic), SHIFT_JIS (a Japanese encoding) save bandwidth by representing only slices of the document character set. These encoding are only useful if the entire document contains characters from one particular script only. UTF-8 (Universal Character Set Transformation Format) encoding scheme, is useful if English is used with other languages in the same document. This encoding will represent all English characters with just one byte, stripping off the higher byte of the two-byte unicode value. Characters of other languages do not undergo any change in this encoding. When a document is transferred on Web, the User-Agent at the other end must be informed as to what is the character encoding of this document.

1.2 Text Rendering

A large character set like Unicode, solves the basic problem of text representation. Since, all the characters of different languages have been assigned unique code there is no ambiguity when using multiple languages simultaneously in a document.

Each font in the system corresponds to a script or language. For example, Courier font corresponds to Roman script and has glyphs for all Roman characters. Text is normally transfered on the Web as a sequence of coded characters; each code value corresponding to a single character. To render a particular character on screen, the User-Agent identifies the script to which the character belongs. From the font-script mapping, an appropriate font representing the script is selected. The document itself may specify the font to be selected or a default font corresponding to the script is selected if none is specified. From the glyph-number mapping, the

exact glyph corresponding to the character is identified and rendered on the screen.

Since there are a large number of scripts, it is not possible for every system to have fonts corresponding to each to these scripts. If the web author uses a script 'X' to create a document but the viewer does not have any fonts corresponding to script 'X', then he or she would not be able to view the document. Each user generally has the fonts for the scripts in which he creates and views the documents. However, when a user want to view a document in a language other than the fonts installed on his system, he would not be able to do so. It is not always enough to have fonts for the script in which the document is written. The code set of web author's font must be the same as the code set of the document viewer's font. For instance, if a web author creates a Devnagari document using UNICODE font mapping, and the viewer has fonts for Devnagari language with ASCII mapping , then he would not be able to view the document.

1.3 Motivation

A couple of solutions have been proposed to overcome the problem of rendering a document for which the fonts are not present on the client system.

Dynamic Fonts

The document itself specifies the source of the font from where the fonts are downloaded, and used to render the document on-the-fly.

Embedded Fonts

Fonts are embedded in the document itself.

However, these solutions require some software on client side which can interpret the dynamic or embedded fonts. Typically, these software should be embedded in the User-Agent itself, so that it can render the document appropriately. However, none of the available User-Agents support both dynamic and embedded fonts. As a result, not all clients would be able to view a document with dynamic or embedded fonts. In this thesis, we intend to provide a portable solution for rendering HTML Devnagari documents. The client need not install any software or fonts on his or her system. Rather he should be able to view the document on-the-fly. Though,

we have provided a solution for Devnagari documents, the same principle can be applied to render a document in any language.

1.4 Thesis Organisation

In chapter 2 we discuss about the related work in the area of multilingual text representation and rendering. In chapter 3 we have presented the design of our approach. Chapter 4 gives implementation details of the approach. In chapter 5 conclusions and future work are discussed.

Chapter 2

Related Work

The main hindrance in achieving multilingualism on Web is poor standards and protocols in terms of multilingual text representation and rendering. Of late, this has been realized and newer standards and protocols have been proposed and implemented. The newer version(1.1) of Hypertext Transfer Protocol(HTTP) has several new tags especially to aid language negotiation and thereby to achieve multilingualism on Web. HTML, the *lingua franca* of Web, has also been modified to include tags which allow one to specify language and direction attributes of a particular section of text. Cascading Style Sheets (CSS) allows one to specify font properties in the document itself for intelligent matching and synthesis of fonts on the client side. Bitstream has developed a technology called **TrueDoc**, which allows one to download the font dynamically along with the document. Microsoft has developed OpenType fonts which can be embedded in an HTML document.

2.1 Multilingualism in HTTP

HTTP[7, 31] is the network protocol that delivers virtually all files and other data on the World Wide Web, whether they are HTML files, image files, query results, or anything else. A User-Agent is an HTTP client because it sends requests to an HTTP server (Web server), which then sends responses back to the client. Several new tags have been defined in the newer version(1.1) of HTTP to help the User-Agent

identify the correct character encoding and language of the document. Also, the protocol enables the User-Agent to give its preferences regarding character encoding and language.

2.1.1 Accept-Charset

The Accept-Charset request-header field can be used by User-Agent to indicate what character sets are acceptable for the response. This field allows clients capable of understanding more comprehensive or special purpose character sets to signal that capability to a server which is capable of representing documents in those characters sets. Each charset may be given an associated quality value which represents the user's preference for that charset. The default value is q=1.

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

This indicates that iso-8859-5 charset is preferable to client than unicode-1-1. If no Accept-Charset header is present, the default is that any character set is acceptable. Accept-Charset allows the server and the client to negotiate a mutually agreeable character set.

2.1.2 Accept-Language

The Accept-Language request-header field allows the client to specify the set of natural languages that are preferred as a response to the request. Each language may be given an associated quality value which represents an estimate of the user's preference for the language. The quality value defaults to q=1.

```
Accept-Language: da, en;q=0.7
```

This means that the user accepts Danish and English but prefers Danish to English. If a document is available in several languages, then the server uses the Accept-Language request-header field to send the correct version to the client. If no Accept-Language header is present, the default is that any language is acceptable.

2.1.3 Content-Type

The charset parameter of Content-Type header field indicates the character encoding of the document sent to the client.

```
Content-Type: text/html; charset=ISO-8859-4
```

The charset parameter helps the User-Agent in selecting an appropriate font to render the document.

2.1.4 Content-Language

The Content-Language response-header field describes the natural language(s) of the intended audience for the enclosed entity. The primary purpose of Content-Language is to allow a user to identify and differentiate entities according to the user's own preferred language.

```
Content-Language: da
```

This mean that body content is intended only for a Danish-literate audience.

2.1.5 Language Negotiation

HTTP 1.1 allows web authors to put multiple versions of the same document under a single URL. Transparent content negotiation[37] is a mechanism, layered on top of HTTP, for automatically selecting the best variant when the URL is accessed. Language negotiation is one of the the facets of content negotiation. A document can exist in several language variants. Using appropriate request header fields, the client indicates its capabilities and/or preferences, which the server then uses to choose the most appropriate version from among all the variants.

- *Server Driven Negotiation*

  - The client requests for a document and sends a preference language list in the request header.

- From the preference language list and the available variants, the server computes the best linguistic version and sends it to the client. A list of available linguistic versions is also sent in the response header.

Disadvantages

- It is impossible for the server to accurately determine what might be "best" for any given user, since that would require complete knowledge of both the capabilities of the User-Agent and the intended use for the response.

- Having the User-Agent describe its capabilities in every request can be very inefficient.

- *Agent Driven Negotiation*

The user-agent first sends a request for the document to the server. The server responds with the list of available variants for the document. The user-agent then sends the request for a specific variant of the document.

Disadvantage is that a second request is needed to obtain the best alternate variant.

2.2 Multilingualism in HTML

2.2.1 FONT FACE

The FACE extension to the HTML FONT tag allows one to add font support to a passage of text.

```
<FONT FACE = "TimesRoman, Courier, Hevletica">

some text goes here

</FONT>
```

The FACE attribute relies on at least one of the fonts specified being installed on the client's system. If the first choice of font is not present on the client's system then the second choice font will be used. If none of the listed fonts are present then the user's own default font will be used.

Disadvantages[18]

- Document would not be rendered properly if none of the listed fonts are present on the client system.
- HTML is about enriching text with hyperlinks and embellishments, but the underlying plain text should already have its proper meaning before HTML markup comes into play. FONT FACE prevents that making searching and sorting an impossible task.
- It is not an official HTML tag

2.2.2 LANG and DIR attributes

For correct display and other operations, it is sometimes necessary to explicitly set the language of a text fragment. Language information can be used to control rendering of a marked up document in a various ways. Some situations where this information helps include:

- Assisting search engines
- Speech synthesis
- Selecting glyph variants for high-quality typography
- Resolving hyphenation ligatures, and spacing
- Spell checking and grammar checking.

Since any text can logically be assigned a language, almost all HTML elements admit the LANG[4] attribute. The language attribute, LANG, takes as its value a language tag that identifies a natural language spoken, written, or otherwise conveyed by human beings for communication of information to other human beings.

The DIR[4] attribute allows web authors to specify the default direction of pieces of text or the text in the entire document.

```
<P lang="en" dir="ltr">...left-to-right english text...</P>

<Q lang="he" dir="rtl"> ...right-to-left hebrew quotation ... </Q>
```

2.2.3 Cascading Style Sheets

The CSS1 specification[8] defines the following font properties for font matching:

- font-family
- font-style
- font-variant
- font-weight
- font-size

Font matching is done using an existing, accessible font that has the same family name as the requested font. The matching information is restricted to the CSS1 font properties, including the family name.

2.3 WebFonts

In CSS1, fonts are assumed to be present on the client system and are identified solely by name. Several choices of fonts may be listed and are tried in order until a font is found that can display the required characters. If a font is available to the client that is a close stylistic match to the requested font but has a different name, it is not possible for a CSS1 implementation to select it.

WebFonts specification[9] extends the font mechanisms in CSS1 and permits improved client-side matching of fonts, enables font synthesis and progressive rendering, and enables fonts to be downloaded over the Web.

- Intelligent matching
  This involves using an existing, accessible font that is the closest match in appearance to the requested font. The matching information includes information about the kind of font, nature of serifs, weight, cap height, x height, ascent, descent, slant, etc.

- Synthesis
  This involves creating a font which is not only a close match in appearance, but also matches the metrics of the requested font. The synthesizing information includes the matching information and typically requires more accurate values for the parameters than are used for some matching schemes. In particular, synthesis requires accurate width metrics and character to glyph substitution and position information if all the layout characteristics of the specified font are to be preserved.

- Download
  This involves fetching a font over the Web and using it to display text in the current document.

- Progressive rendering
  Progressive rendering is a combination of download and one of the other methods; it provides a temporary substitute font (using name matching, intelligent matching, or synthesis) to allow content to be read while the requested font downloads. Once the real font has been successfully downloaded and temporarily installed, it replaces the temporary font.

2.4 TrueDoc and Dynamic Font

Dynamic fonts[5] allows the web authors to specify exactly the font(s) that are acceptable in a given piece of text. Authors can supply a font, guarantying complete control of the typeface by placing a link to an external font resource. This tells the client to follow the accompanying link to dynamically download the font with the page.

TrueDoc[2, 1] is a technology developed by Bitstream for the creation, transport . and imaging of platform independent scalable font objects on the Web. TrueDoc's character shape recorder (CSR) is used to create font objects and TrueDoc's character shape player (CSP) is used to render the font objects.

2.4.1 Character Shape Recorder

The CSR records character shapes from the fonts that web authors use in their documents and stores them in a highly compact data structure called Portable Font Resource (PFR). Input to CSR may be either TrueType or Type1 of any flavor on either Windows, Mac or Unix. When recording characters, the CSR does not access the original font directly. Rather, it depends upon the native font engine, such as Adobe Type Manager or a TrueType rasterizer in the Macintosh or Windows operating system, to process the original font program. The CSR can record any character shape, including symbols, and is fully compatible with double-byte fonts using any encoding. CSR uses character subsetting. It records shapes of only those characters which have been used in the document; the entire character set of the font is not included. This is especially important for Asian languages, where a complete character set includes thousands of glyphs. Typically, the CSR is integrated with the authoring tool used to create web documents.

2.4.2 Character Shape Player

The CSP reads the PFR at the receiving end and plays back the character shapes that the CSR recorded. The CSP intelligently scales and rasterizes the character shapes in the document at the required point size and resolution. Since the PFR contains scalable fonts, user can zoom in and out of the document; the character shapes redraw on the fly. For enhanced performance, CSP uses a high-speed bitmap cache. CSP generates standard font formats at the receiving end, so that they can work seamlessly with existing operating systems and printers. The generated fonts are installed only temporarily for the purpose of rendering and viewing the original document. Typically, CSP would be integrated in a web browser.

2.4.3 Portable Font Resource

PFR datafile is output by the CSR. It is a highly compressed, resolution independent representation of the shapes of all the characters in the document. It contains scalable outline information of all the different characters in one compressed file. The size of the PFR varies according to the number of different character shapes in the recorded document. PFR offers 2-to-1 compression over PostScript fonts, and better than 3-to-1 compression over TrueType fonts. This high compression is achieved due to character subsetting, sophisticated character recognition to break compound characters down into their component parts, and using a cubic bezier format to store the recorded character shapes. Since, the PFR is separate from the text, browsers that can't read PFR's do not lose any of the content of the page. Also any text formatted with PFR's is searchable, retaining the qualities of any HTML text. The outline information in the PFR is encrypted to prevent piracy.

2.4.4 Specifying Dynamic Fonts in HTML Documents

Dynamic fonts can be included in HTML document in two ways[30]:

- FONT FACE tags is used within the HTML document to specify the fonts, and a link tag is used to associate the HTML document with the PFR that the TrueDoc enabled authoring tool generates.

  ```
  <HEAD><LINK REL="fontdef" SRC="mypage.pfr"></HEAD>
  ```

  The REL attribute specifies a font definition, while the SRC attribute specifies the source for the font definition.

- Cascading Style Sheet can be used to specify the fonts by including the URL for the source of the font data.

  ```
  <HEAD><STYLE TYPE=ntext/cssn>
  @font-face {font-family: Gothic;
  ```

```
src: url(http://mysite/path/mypage.pfr);
     font-weight: bold} </HEAD>
```

2.4.5 Overall Architecture

Using a TrueDoc enabled authoring tool, web author creates an HTML document and the corresponding PFR file. The PFR file resides on the server along with the document. When a TrueDoc enabled browser sends request for the document, the PFR file is dynamically downloaded along with the document. The browser then uses the PFR file to render the document.

Advantages

- Since the font information in the form of PFR is downloaded along with the document. the browser can render the document even if the requested fonts are not present on the client system.

Disadvantages

- Additional overhead of downloading the PFR.
- Character Shape Player must be present on the client system. Browsers which do not have CSP built into them would not be able to render the document as intended by the web author. As of now, only Netscape Communicator has CSP built into it.

2.5 OpenType and Embedded Fonts

2.5.1 Font Embedding

Font embedding[11] is a method of including fonts with the documents in which they are used. By embedding a font within a document, web authors ensure that their text will appear correctly on other client systems, even if that system doesn't possess the correct fonts. Because the fonts are included with the actual document,

the user's operating system and application or viewer is able to access the embedded fonts and make use of them. In contrast with only copying selected elements of a font, embedding the font involves sending the real font, with all of its original high quality outlines, metrics and hinting information.

Levels of Embeddabililty

There are three levels of embeddability used in fonts which allow a font creator to control and restrict the use of the font according to their wishes.

Fully installable means that the font contained in the document can be installed on the client's system and used without restriction in other application.

Editable means that the font can be used to make changes to the document in which the font is embedded, but cannot be used elsewhere.

Do not embed means that the font cannot travel with any document.

Subsetting

Because a document rarely uses all of the characters contained in a font, the size of the document can be reduced by subsetting the font. Subsetting allows only the characters used in a text to be sent with the document. Subsetting a font means that the font is no longer editable by the recipient.

2.5.2 OpenType Fonts

OpenType[12, 28, 29] is an extension to the TrueType font format. It contains additional information that extends the capabilities of the fonts to support high-quality international typography. OpenType can associate a single character with multiple glyph representation, and combinations of characters with a single glyph representation. OpenType includes two-dimensional information to support features for complex positioning and glyph attachment.

- *Features*

  - Advanced typographic support - including ligatures, small caps, old style numerals, alternate letterforms, and contextual substitution.

  - Support for the Unicode character standard

  - Multiscript character sets

  - Helps an application use the correct forms of characters when text is positioned vertically instead of horizontally.

  - Contains baseline information that specifies how to position glyphs horizontally or vertically. Because baselines may vary form one script to another, this information is useful for aligning text that mixes glyphs from scripts for different languages.

  - Better protection of font data by using industry-standard digital signature technology.

OpenType fonts can be embedded in web documents using Microsoft Web Embedding Font Tool (WEFT)[21]. At the receiving end, browser must be capable of understanding embedded OpenType fonts. The subsetting and compression technology of OpenType make it relevant to the Internet and the World Wide Web, since it allows for fast download of type.

Advantages

- Since the fonts are embedded in the document itself, web authors can guarantee that their documents and files will appear exactly as they intended them to.

Disadvantages

- Additional overhead of sending font information along with the document.

- OpenType font format is platform dependent. Hence, it is not possible for clients on all the platforms to view the document.

- A browser which can understand embedded OpenType font format is needed on the client side. As of now, only Internet Explorer (4.0) supports OpenType font embedding.

2.6 Portable Documents

2.6.1 JetSuite, Digital Paper, and Envoy 7

JetSuite

Any formatted document can be converted to Jetsuite[14] Document (*jsd*) by using JetSuite printer driver. *jsd* is a highly compressed file containing information about fonts, color and formatting. *jsd* file can be viewed with a JetSuite viewer. Documents saved out in *jsd* format can be transmitted anywhere and viewed on the destination computer exactly as the web author intended them to be seen. The JetSuite viewer can be embedded in the file itself to create a self-viewing file. The embedded viewer can be detached from the original file for later use by the client. JetSuite also allows to link to documents on the Internet by placing hyperlinks in the documents.

Advantages

- Since fonts and formatting information is embedded in the document itself, the web author can ensure that document is presented to the viewer in exactly the same form that he intends, no matter what fonts are installed on the client system.

Disadvantages

- Overhead of sending fonts and formatting information along with the document.
- Self-viewing JetSuite documents have the additional overhead of JetSuite viewer.
- Clients need to have a JetSuite viewer installed on their system if the document is not self-viewing.

- The only platforms supported are windows 3.1 and Windows 95

Digital Paper [38] and Envoy 7 [22] are similar to Jetsuite except that they provide the viewer as a browser plug-in also, enabling the document to be viewed on any platform.

2.7 Table of Comparisons

A comparison of the above technologies in terms of features and overhead is given below.

2.7.1 Feature Comparison

	Portability	Browser Support	Font
TrueDoc	Windows. Unix, Mac	Netscape 4.01	Any
OpenType	Win 95, Win NT	Internet Explorer 4.0	OpenType
Portable Document (Digital Paper)	Windows, Unix, Mac	Viewer/Pluggin required	Any
Embedded Font (MS Word)	Win 95, Win NT	Viewer required	Any font whic allows Embeddi

2.7.2 Overhead Comparison

The technologies discussed in the previous sections allow the user to view a document whose font is not supported on the client system. However, this imposes an additional overhead of sending the font (and formatting) information along with the

document to the client side. The overhead incurred on documents of sizes 1k, 10k, and 40k is shown below.

	1k	**10k**	**40k**
TrueDoc	13k	37k	30k
Portable Document (Digital Paper)	20k	71k	160k
Embedded Font (MS Word)	12k	18k	39k

The overhead is dependent on whether the formatting information is included along with the document and the number of different font faces used in the document. For instance, Portable Document and Embedded Font has more overhead than TrueDoc since they include the formatting information along with the document. A document which uses three different font faces will have more overhead than the one which uses just a single font face for the entire document.

Chapter 3

The Design

The main problem in allowing a client to view a multilingual document is the lack of appropriate fonts on the client system. Traditional solutions require the client to download and install either the fonts or some software on the system to aid in the process. However, these solutions have the inherent disadvantages of security risks, disk space usage, and non-portability. In some cases, it may also violate the intellectual property rights(IPR) of the font manufacturer or the software developer. Other solutions which permit dynamic downloading of fonts to render the document are browser dependent. For a truly portable, secure, disk space-efficient, browser-independent solution, the client should not download and install any software or fonts on the system. Rather, (s)he should be able to view the document in a transparent manner.

In this chapter, a design for one such solution is presented.

3.1 The Basic Approach

The basic approach is to make the server send the document, the required fonts, and a handler program which will render the document on the client system using the accompanying fonts.

3.1.1 Font Handler Applet

The handler program should have the following basic characteristics:

- It should be portable and it should be able to render the document on a variety of platforms and operating systems.
- Since the handler program is executed on the client side, it should be secure.
- The handler program, transmitted across the network to the client side, should be small so as to save network bandwidth.
- It should be efficient. The client should not wait too long for the program to render the document on the screen.
- It should be able to understand the format in which the font information is passed along with the document.
- Since. the document is in HTML form, all the interpretation of HTML tags and the corresponding formatting should be done by the handler program.

The choice of Java

Java is a platform- and operating system-independent programming language. A Java applet is a program which is downloaded across the network and then interpreted on the client side by a Java enabled browser. Since, Java is inherently secure, applets written in Java do not pose any security threats to the client system. Portability and security features of Java make it an ideal programming language for writing the handler program. The handler program is, therefore, written as a Java applet.

3.1.2 Fonts

Since it is assumed that the fonts are not present on the client system, the font information must be passed along with the document. Typically, font information

will be downloaded by the handler program from the server and used to render the document. This font information can be provided to the applet in 3 possible ways:

- The same font file which the web author uses to create the document can be downloaded by the applet to render the document.
  Disadvantages:
  - Different web authors may use different font file formats to create web documents. For instance, one may use a TrueType font file while another may use a type1 font file. This means that the applet must be capable of interpreting different font file formats. This will increase the complexity and thereby, the size of the applet. Alternatively, the applet will need to be modified to interpret the required font file format.
  - The standard font file formats like truetype, type1 etc. are complex and use bezier curves to describe a character. This makes the rendering of each individual character on the screen by the applet fairly complex. Also, if the font bitmaps are cached on the disk of the client, since the applet cannot access the disk on the client side, the rendering process becomes slow.
- Using a platform independent font format like *pfr*[2.4.3] to render the document. Advantages:
  - The same applet can be used to render the documents irrespective of the format of the font file used to create the document.
  - Since the fonts are scalable, only one font file needs to be sent to the client to render characters of various sizes.

  Disadvantages:
  - Since a *pfr*-like format is complex, it is difficult to generate the bitmap for the character on-the-fly without using any system resources.

- Send gif images of the glyphs used in the document to the applet for rendering the document.

  Advantages:

  - The applet is relatively simple, since it does not have to interpret complicated font files. This reduces the size of the applet considerably. This also saves on network bandwidth when the applet is transmitted from server to client.
  - Gif images are platform independent. The same applet can be used to render the document irrespective of the format of the font file used to create the document.
  - The time taken to render the document is considerably less since the glyphs need not be generated at the client side.

  Disadvantages:

  - Since the gif images of characters are not scalable, separate gif images corresponding to each of the different sizes of the characters used in the document need to be sent to the applet.

After considering the merits and demerits of the various approaches of sending font information to the client, the approach of sending font information as gif images is chosen in this design.

3.2 Server Script

The primary job of the server script is to send only those character-images which are not present on the client side. The client applet sends the file name of the document and a list of numbers indicating character-images present on the client side to the server script. The script parses the HTML document, interprets various HTML tags and makes a list of the various characters with their corresponding sizes and styles, used in the document. The image of each character is stored in a separate file. In an HTML document, seven sizes and four styles of fonts are permitted. So, for each

character in a character set. 28 different files, each corresponding to a particular size and style is needed. The script concatenates the gif files corresponding to the characters which have been used in the document, into a a single file. The path of this file is sent back to the client. The applet then downloads this file separately.

3.2.1 Other Alternatives

- A Master Image File containing images of all the characters of a character-set can be kept on the server. There will be a separate master image file corresponding to different sizes and styles of characters. The server script will extract the images of characters from the master image files and prepare a new image file containing only those character images which have been used in the document. The path of this new image file will be sent to applet.
  Advantages:
  - Fewer image files need to be stored on the server.

  Disadvantages:
  - Server script becomes complex, thereby, increasing the load on server.
- Another alternative could be to club groups of frequently used characters in one image file. less frequently used characters in another file and so on. In this approach no new image file will be made on-the-fly, but the entire image file will be sent even if just a single character has been used from that image file.
  Advantages:
  - Processing time is reduced since new image files are not made.
  - Number of files to be maintained is less.

  Disadvantages:
  - More overhead on network bandwidth.
- Another approach could be to send Master Image File itself to the client side. Characters from this file will be used by the applet for rendering documents

on subsequent request. There will be no need for the applet on the client side to communicate to the server about what characters are already present at the client. The server needs to send only those new characters that are present in the next document.

Advantages:

- Server script only needs to know different styles and sizes of fonts used in the document. Since the entire master image file is sent, it need not keep track of the characters and their styles and sizes.

Disadvantages:

- For large character sets, the overhead of network bandwidth would be more. Also, the number of Master Image Files to be sent to the client side is dependent on the various sizes of characters used in the document. For instance, for a particular font size and style only 4 different characters are used but the entire Master Image File would still need to be sent.

This approach is ideal for situations where small character-sets are used and same font size and style are used in entire document.

The script also sends font metric information like, the width of the characters used in the document, to the client applet.

3.3 Client Applet

The primary job of the applet is to render the document using the downloaded gif images of the characters. The applet will have as its parameters the path of the document to be rendered, the path of the server script, and a list of height values giving information about various sizes of characters permitted in the document. The applet will send the path of the document to the server script, which will then return the path of the file containing images of characters used in the document. The script will also return font metrics information like width of the characters used in the document. The applet will then download the image file, extract individual

images from it and store it in a data structure. The applet also needs to download the actual document.

Once the applet has the document, the character images and font metric information it begins the process of parsing the HTML document. The applet interprets all the HTML tags and performs the corresponding formatting. Since, the rendering of the document is done by the applet itself, all the formatting will also have to be done by applet itself. In this sense, the applet has some of the functionality of a browser.

Basically, the applet uses the character images and prepares an image of the entire document. The HTML tags are interpreted and the character images placed in such a way as to give a look-and-feel effect, as if they have been rendered by the browser itself. The applet needs to keep track of font size and style changes, so that the appropriate image of the character is used.

The font metrics information like the height and width of characters is used to place the character images appropriately. It gives information as to how much to advance horizontally after placing an image for a character and how much to advance vertically in case of a new line. Care must be taken to adjust the baseline when different sizes of characters are used in the same line. Also, language specific information , if any, must be encoded in the applet to render the document properly. Typically, this information includes directionality, ligatures, overhangs, underhangs, and the like .

An HTML document may contain hyperlinks to other documents on the same server or on some other server. This calls for the applet to keep track of mouse events, so as to take action when a user clicks on a link. A Java applet can only communicate with the host from which it got loaded. For this reason, a new browser window is spawned up in case the user clicks on the link which points to document residing on the server other than the one from which the applet got downloaded. The URL of the document is submitted to this new browser window, which then downloads and renders the document. However, if the link is to the document on the same server as the applet, then the applet wipes off the image of old document, downloads the new document and the character images, constructs an image for the

new document and renders it in the same browser window.

3.4 Optimizations

To reduce the network bandwidth utilization and to speed up the computation at the server end, certain optimizations are done:

- When the applet reaches on the client and starts functioning, it downloads the document and the character images. Later if the applet sends a request for another document, it should not download the previously downloaded character images again. Therefore, the applet sends a list of numbers to the server script indicating which images are present on the client system, so that these need not be sent again.

- The server script computes the list of character images required in the requested document and also the associated font metric information like the width of different characters used in the document. To reduce the time taken to compute this information, the script stores this information in a file when the document is requested for the first time. For, each subsequent request for the same document, the script gathers the required information from the stored file.

- The size of the gif image of a character is reduced from about 450 bytes to about 125 bytes. by removing the Comments part of the header from it.

- The gif images of characters are concatenated into a single file so that the applet can download all the images in a single TCP/IP connection.

Chapter 4

Implementation

In this chapter, the implementation for the dynamic display of unsupported fonts is presented. This includes details about the client side Java applet, server side Perl script, and the implementation of other programs on the server side to generate fontmetric information, club character-images etc. are discussed.

It is assumed that the user will initially access the home-page and then access other pages through the links present in the home-page. The user is not allowed to directly specify the URL in the browser of any other file on the server.

4.1 Client Applet

4.1.1 Major Classes

Major classes used in the implementation of the Java applet are as follows:

Parser class takes as input an HTML file, and parses it for HTML tags. It produces an instance of Document class as a result.

Document It is a class which holds an HTML document. An HTML document consists of parsed text, a vector of tags and optional HTML source text. Document is used by Parser to store parsed documents.

Tag class is used to store properties of predefined HTML tags like name, id number, whether it is an inline or block level tag, and whether it has a corresponding

end tag or not.

TagRef class gives the reference of the tag, its attributes, and its position in the actual text.

FontAplt This is the main class and includes functions for downloading the document, communicating with the server script, getting the character-images and fontmetric information, and finally rendering the document on the client's screen.

The path of the document and the server script is passed to the applet as parameters. The applet establishes connection with the server script and passes it the path of the document. The server script returns the path of the image file containing character-images, fontmetric information, and other information needed to extract individual character-images from the image file. The FontAplt then calls the Parser for parsing the input HTML document. The parsed output is returned to the FontAplt as an instance of the Document class. The vector of HTML tags is then looked up one by one based on which various actions are taken as to how the character-images should be placed. The HTML tags supported by the FontAplt are :

```
<P align = left|right|center|justify>
<H1 align = left|center|right> to <H6>
<FONT size = [+|-]number>
<EM>
<STRONG>
<B>
<I>
<CENTER>
<BR>
<A href = url>
```

ScrollableCanvas The size of the applet canvas is fixed. But, the document to be rendered may be of any size. If the image of the document does not fit in the canvas, scrollbars are provided so that the user can scroll the document. This class paints the visible portion of the image depending on the position of the horizontal and vertical scrollbars.

Format Depending on the HTML tag being processed, this class sets various flags to enable correct formating of the document.

TagProcessor Processes the HTML tag.

Href class is used to store information about a hyperlink, like the area of image where this hyperlink is mapped and its corresponding URL.

4.1.2 Processing of HTML tags

<P> Depending on what align attributes are set, the images are placed in a manner so as to give an effect of left, right, center aligned, or justified text. The default alignment is *left* in case no align attribute is specified. The images of the characters must match the current font size and style.

<H> In this case also, the align attributes determine as to how the corresponding text should be formatted.The default alignment is *left*. Font style is changed to bold and font size is set according to the type of the header tag.

<FONT> Depending on the size attribute, it is determined whether the font size is to be set to an absolute value or relative to the base font size. The default value of font as well as base font size is 3. A stack of font sizes is maintained since the *FONT* tag may nested within other *FONT* tags. When a start *FONT* tag is encountered, the font size is pushed on the stack and when an end *FONT* tag is encountered, the font size is popped from the stack. The current font size is the one on top of the stack.

<I>/<EM> The italics flag is set or reset depending on whether it is a start or an end tag. If the italics flag is set, the font style is changed to italics or

bold-italics depending on whether the bold flag is set or not.

<B>/<STRONG> The effect is same as in case of *<I>* tag except that the bold flag is set or reset.

<CENTER> The corresponding text is centered.

**
** This tag causes a line break.

<A> When an anchor tag is encountered, the position of the text to be anchored and the corresponding URL is noted. The text is underlined in red color so as indicate an hyperlink. The mouse events are tracked and when a user clicks on to the hyperlink, the new document is rendered either in the same window or in a new browser depending on whether the URL is of the same site as the applet or not. In case, the URL is of the same site as the applet, the server script is contacted and the new document as well as the character-images are downloaded. The image of the old document is wiped off and the image of the new document is rendered. The applet also sends to the server script a list of numbers indicating which character-images are present on the client side so that they need not be sent again. However, if the URL is not of the same site as the applet, a new browser window is spawned up and the URL is submitted to it.

4.2 Server Script

The server script is coded in Perl. Since Perl is platform- and operating system-independent, the script need not be rewritten when ported to a different environment. The script takes as input the path of the document and a list of numbers indicating character-images present on the client side. The script then parses the HTML document. The HTML tags like *H*,*FONT*, *B* and *I* which affect the font size and style of the characters are tracked. The script makes a list of characters with their corresponding font sizes and styles. This list is stored in a file so as to avoid recomputation when this document is requested again. From this list and the

list which the applet sent, a new list is prepared which contains only those characters which are not present on the client side. This new list is submitted to a Java program which clubs the individual character images into a single image file. The new image file name is returned to the script which then sends it back to the applet. Information to enable the applet to extract the individual character-images from the image file is also sent. The script also sends the fontmetric information corresponding to the characters in the list to the applet.

4.3 Generating Fontmetric Information

The fontmetric information is required by the client applet for correct placing of character images. This information includes character widths and the font height in pixels. The character width specifies the exact horizontal advancement to be made before and after placing the character-image. The font height specifies the exact vertical advancement to be made for a newline. Fontmetrics information can be generated by Java. But, Java allows this information to be generated for only Java fonts. Since, Devnagari fonts are not in the list of fonts supported by Java, this information is generated by a native C program using Windows API system calls. This is the only program in the system which is platform-dependent. The input to the program is the font name for which the fontmetric information is to be generated. The fontmetric information for the required font styles and sizes are generated and stored in corresponding files. These files are used by the Perl script to send the fontmetric information to the client applet. The generation of fontmetric information files is a one-time process.

4.4 Image Clubbing

Each character image is stored in a separate file. If each character-image file is downloaded separately, the applet would have to open as many TCP/IP connections to the server as the number of images to be downloaded. This would increase the time taken to download the images, as well as increase the load on the network, the client

side CPU and the server side CPU. For this reason, the images are concatenated so as to form a single image file. This is done by invoking a Java program from the Perl script and passing it the list of image files to be clubbed. The applet at the client side extracts individual character-images from this file.

4.5 Nomenclature

As discussed earlier, each character image is stored in separate file. In an HTML document, four font styles - normal, italics, bold and bold-italics, and seven font sizes can be used. If the character set has 100 different characters, then 2800 image files will be needed to store character images. To identify each of these files uniquely the following nomenclature is adopted:

```
character-code.font-size.font-style.extension
```

The following codes have been assigned to font-styles:

```
normal       - 1
italics      - 2
bold         -  3
bold-italics - 4
```

The image file of a character of code 100, font size 3, font style bold and stored as gif will be named as 100.3.3.gif.

4.6 Overhead

The additional overhead of sending the document to the client includes the applet and the character-images. The number of different gif images to be sent to the client is dependent on the different styles and sizes of characters used in the document. For a particular session, the applet needs to be sent only once. Only those character-images are sent to the client that have not been sent previously in a particular session, thereby, reducing the number of gif images transmitted to the client on subsequent requests.

The size of the applet is 58k and the size of each character-image is about 125 bytes.

For instance, when a user first connects to the site, the document, the applet and character-images are transmitted to the client side. Assuming a document of size 10k using around 100 different characters with different sizes and styles, the overhead of sending the applet and the gif images is around 70k. However, when the user requests for another similar sized document in the same session, which uses 20 characters different from the previous document, the overhead is just around 2.5k. This is because the applet and the already downloaded character-images are not downloaded again.

Chapter 5

Conclusions and Future Work

In this report. we have presented a solution that enables a client to view a web document that uses a font not supported on the client system. The solution involving the use of Java applet and Perl scripts is secure, portable and transparent to the end-user. The solution was implemented for *Devnagari* script using a font ("*Bhagwan*") not normally supported by general purpose systems. With this implementation, a user can visit a website that publishes its documents in Devnagari and view them through a browser. This is, provided the server script and helper utilities are installed on the server side. The solution can be extended to other fonts just by providing the appropriate glyphs on the server side. This way the server can be made to serve a multi-lingual website without the need to modify the server.

Devnagari script was chosen for implementation because documents in Hindi, Marathi, Sanskrit can be published using it. The solution could be used by newspaper publishers who currently require the users to download the fonts before viewing the documents in their web sites. As more and more of multi-lingual India accesses the World-Wide Web, this solution could provide an answer to many website owners.

The work done as of date has the following limitations.

5.1 Limitations

1. Since the gif images of characters are not scalable, several gif images for the

same character, corresponding to different font sizes and styles are required to be sent.

2. Since the handler program is written as a Java applet, the client browser must be Java enabled to view the document.

3. Since the document is shown as an image in the applet's window, the user cannot save the document for offline viewing.

4. The user cannot edit the document.

5. Text processing operations like searching, sorting etc. cannot be performed on the document.

6. When accessing documents on the server, the user has to start browsing from the homepage.

7. For each font-face which a web author uses to create documents, the corresponding gif images of characters in that font-face need to be generated.

8. Since each gif image is stored in a separate file, a large number of files is to be maintained on the server.

5.2 Future Work

1. The applet can be modified to interpret a platform-independent font format like *pfr* [2.4.3]. This would eliminate the need for creating and sending the gif images of characters.

2. At present the applet shows documents which homogeneously use one particular font only. However, it can be extended to show a document containing parts shown in more than one font.

3. Threading mechanism in Java can be exploited to render the document as it is being downloaded. Also, the image-file as well as the document can be downloaded parallely.

4. The applet can be extended to add support for HTML tags like *lists*, *tables*, *forms*, *frames*.

5. If the client supports Devnagari script then the server need not send the applet. Instead the actual document may directly be sent to the client. Language negotiation feature of HTTP 1.1 can be used to find whether the required character-sets are present on the client side.

6. Cookie mechanism can be used to store the gif images on the client side.

7. To save network bandwidth, the document can be compressed before sending and again decompressed at the client side.

5.3 Comparison With Other Technologies

5.3.1 Feature Comparison

	Portability	Browser Support	Font
TrueDoc	Windows, Unix, Mac	Netscape 4.01	Any
OpenType	Win 95, Win NT	Internet Explorer 4.0	OpenType
Portable Document (Digital Paper)	Windows, Unix, Mac	Viewer/Pluggin required	Any
Embedded Font (MS Word)	Win 95, Win NT	Viewer required	Any font whic allows Embeddi
Our Solution	All Platforms	Any Java enabled Browser	Any

5.3.2 Overhead Comparison

In case of other technologies discussed in chapter 2, the font information is sent with every document. The browsers have no way of informing the server about the font information present on the client side and also since the HTTP servers are stateless, they cannot keep track as to what font information have been sent to which client. For this reason, the font information is required to be sent with every document. Also, the web author, while creating the document links or embeds the font information along with the document. Each document is treated as a separate entity with all the required font information.

In our solution, the Java applet and the Perl script negotiates with each other and only those character-images are sent to the client which are not present on the client side. This greatly reduces the amount of extra font information required to be sent to the client on subsequent requests. The applet is also sent only once for a particular session. The solution is ideal for situations where a user connects to the site and views a number of pages in succession:

Glossary

Character A single graphic symbol represented by sequence of one or more bytes.

Character Encoding Scheme The mapping from a coded character set to an encoding which may be more suitable for specific purpose. For example, UTF-8 is a character encoding scheme for ISO 10646.

Character Set An enumerated group of symbols (e.g., letters, numbers or glyphs)

Coded Character Set The mapping from a set of integers to the characters of a character set. For example unicode, ISO-8859-1

Document Character Set is a set of abstract characters and a corresponding set of integer references to those characters. SGML considers a document to be a sequence of references in the document character set.

Font is a collection of glyph representations having the same basic design according to design, size, appearance, writing system, and other attributes associated with the entire set.

Glyph Graphical representation of character.

Bibliography

[1] TrueDoc FAQs.
http://www.bitstream.com/tdfaq.htm

[2] TrueDoc Font Technology - Overview.
http://www.bitstream.com/tdwp.htm

[3] Create your own multilingual Web site.
http://www.alis.com

[4] HTML 4.0 Specification.
http://www.w3.org/TR/WD-html40-970708/

[5] Dynamic Fonts.
http://www.bitstream.com/dynamic.htm

[6] Internationalization/Localization: HTML.
http://www.w3.org/International/O-HTML.html

[7] Internationalization/Localization: HTTP.
http://www.w3.org/International/O-HTTP.html

[8] Cascading Style Sheets, level 1.
http://www.w3.org/pub/WWW/TR/REC-CSS1

[9] W3C Working Draft: Web Fonts.
http://www.w3.org/TR/WD-font

[10] M.T. Carrasco Benitez. Web Internationalization and Multilinguism.
http://www.w3.org/International/tomas.carrasco.benitez.html

[11] Embedding TrueType.
http://www.microsoft.com/typography/embed/embed.htm

[12] OpenType Specification.
http://www.mircosoft.com/typography/otspec/otsp101.zip

[13] TrueType Specification.
http://microsoft.com/typography/tt/ttf_spec/ttspec.zip

[14] Jetsuite: Product Specs: Jetsuite Viewer.
http://www.west.net/ dfi/jetsuite/jsviewer.html

[15] i18n/l10n: HTML - multilingualism.
http://www.w3.org/International/O-HTML-multilingual.html

[16] Unicode: Basic Principles.
http://www.unicode.org/unicode/standard/principles.html

[17] F. Yergeau, G. Nicol, et. al.. RFC 2070 - Internationalization of HTML.
http://www.alis.com:8085/ietf/html/rfc2070.txt

[18] <FONT FACE> considered harmful.
http://www.isoc.org:8080/web_ml/html/fontface.html

[19] Character encoding.
http://www.isoc.org:8080/codage/index.html

[20] Scientific American: Article: Multilingualism on the Internet.
http://www.sciam.com/0397/issue/0397oudet.html

[21] Mircosoft Web Embedding Fonts Tool.
http://www.microsoft.com/typograph/web/embedding/weft/default.htm

[22] Tumbleweed Software - Products.
http://www.twcorp.com/viewer.htm

[23] Specifying fonts in Web pages.
http://www.microsoft/com/typography/web/designer/font-face.htm

[24] Francois Yergeau. A world-wide World Wide Web.
http://www13.w3.org/International/francois.yergeau.html

[25] The Unicode Standard.
http://www.unicode.org/unicode/standard.html

[26] Carrasco Benitez. Web Internationalization.
http://www.dragoman.org/winter/draft0.html

[27] OpenType Jamboree.
http://www.microsoft.com/typography/jamboree/default.htm

[28] The OpenType Font File.
http://www.microsoft.com/typography/otff.htm

[29] OpenType FAQ.
http://www.microsoft.com/typography/faq/faq9.htm

[30] Including dynamic fonts in HTML documents.
http://hills.ccsf.cc.ca.us:9878/ awilso01/project.html

[31] R. Fielding, et. al.. Hypertext Transfer Protocol - HTTP/1.1.
RFC 2068

[32] International HTTP.
http://www.isoc.org:8080/web_ml/http/index.html

[33] Internationalization of HTML.
http://www.alis.com:8085/ietf/html/

[34] i18n/l10n: fonts.
http://www.w3.org/International/O-fonts.html

[35] Thomas Boutell. CGI programming in C and Perl.
Addison-Wesley Publishing Company.

[36] Patrick Naughton, Herbert Schildt. Java: The Complete Reference.
Tata McGraw-Hill Publishing Company.

[37] Transparent Content Negotiation in HTTP.
http://gewis.win.tue.nl/ koen/conneg/ .

[38] Digital Paper.
http://www.hummingbird.com/whites/index.html

[39] Larry Wall, Randal L. Schwartz. Programming Perl.
O'Reilly & Associates, Inc.